Hardware Verification Using PVS *

Mandayam Srivas, Harald Rueß, and David Cyrluk

1. Introduction

The past decade has seen tremendous progress in the application of formal methods for hardware design and verification. Much of the early work was on applying proof checking and theorem proving tools to the modeling and verification of hardware designs [Gord83b, Hunt89]. Though these approaches were quite general, the verification process required a significant amount of human input. More recently, there has been a large body of work devoted to the use of model checking, language containment, and reachability analysis to finite-state machine models of hardware [ClGr87a, BCMD92, BCLM94]. The latter class of systems work automatically but they do not yet scale up efficiently to realistic hardware designs. The challenge then is to combine the generality of theorem proving with an acceptable level of effective and efficient automation.

Our main thesis is that in order to achieve a balance between generality, automation, and efficiency, a verification system must provide powerful and efficient primitive inference procedures that can be combined by means of user-defined, general-purpose, high-level proof strategies. This design philosophy has formed the guiding principle for the implementation of the PVS system [OwRS92, ORRS96, ORSH95]. It combines an expressive specification language with an interactive proof checker that has a reasonable amount of theorem proving capabilities. PVS is designed to automate the tedious and obvious low-level inferences while allowing the user to control the proof construction at a meaningful level. Exploratory proofs are usually carried out at a level close to the primitive inference steps, but greater automation is achieved by defining high-level proof strategies. When compared to other proof checkers, the primitive inference steps of PVS are very powerful as they are implemented using a set of powerful decision procedures.

The domain of problems that have been investigated with PVS involves verification of industrial-strength microprocessors [MiSr95, SrMi95a, Cyrl96], protocol verification [Hoom95, HaSh96, PaDi96a, Shan92], arithmetic circuits [RuSS96, Rues96, MiLe96], real-time properties [Shan93, Hoom94], fault tolerance [ViBu92, Mine93, Rush93], and clock synchronization [Shan92, Rush94, MiJo96].

* The development of PVS was funded by SRI International through IR&D funds. Various applications and customizations have been funded by NSF Grant CCR-930044, NASA, ARPA contract A721, and NRL contract N00015-92-C-2177.

Although PVS is a general purpose theorem prover, it supports the specific needs of hardware verification through the use of an expressive specification language, a bit-vector library, decision procedures for equality, linear arithmetic, and arrays, propositional simplification based on binary decision diagrams, and integration of symbolic model checking.

While PVS is capable of verifying a wide variety of hardware circuit designs, most of the large verifications we have performed are in the area of pipelined microprocessors and complex arithmetic circuits at register transfer level. The reason we have concentrated our effort on datapath-intensive circuits at register transfer level is because theorem-proving techniques are most effective in these domains. Also, the inadequacy of conventional simulation-based CAD tools is most pronounced at register transfer levels and higher for complicated designs involving, for example, pipelining. So, we will devote most of this chapter to describing the approaches to verification in these domains of applications.

This chapter is organized as follows. Sect. 2. contains a comparison of PVS with related theorem proving systems, and in Sect. 3. we describe the basic features of the PVS specification language and the PVS prover. Predicative and functional styles of hardware descriptions in PVS are discussed in Sect. 4. In that section we also demonstrate the capabilities of the PVS specification language to model generic hardware components. The next two sections are devoted to specifications, methodologies, and proofs for verifying microprocessors and arithmetic circuits. Sect. 5. includes a description of the basic methodology of processor verification together with the verification of toy processors including the Tamarack processor and a discussion of how these techniques scale up for verifications of industrial-strength processors. Sect. 6. provides a description of a hierarchical verification of a combinational multiplier. In that section, we also outline the verification of an SRT division circuit that is similar to the one in the Pentium microprocessor. Finally, Sect. 7. summarizes the experiments we have performed on verifying the circuits (single pulser, arbiter, Black Jack, FIR filter) used throughout this book.

2. Related Work

The PVS system is engineered by combining a number of theorem proving techniques some of which were pioneered and proven effective in other systems. For example, NUPRL [Cons86] and VERITAS [HaDa92a] provide predicate subtypes and dependent types, and the theorem proving techniques draw on LCF [GoMW79], the Boyer-Moore prover [BoMo79, BoMo88], and on earlier work at SRI [Shos84]. Historically, theorem proving systems have made a trade-off between expressiveness of the logic/specification language supported and the degree of effective automation provided. PVS differs from others in its aggressive use of decision procedures and in tightly integrating